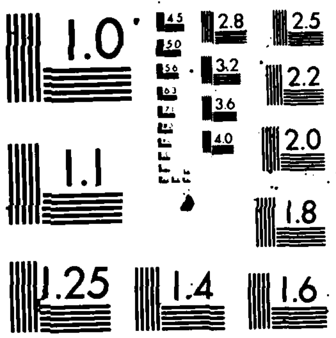
AD-A171 742 METHODOLOGY VERIFICATION OF HIERARCHICALLY DESCRIBED 1/1
VLSI CIRCUITS(U) MASSACHUSETTS INST OF TECH CAMBRIDGE
LAB FOR COMPUTER SCIENCE I L BAIN ET AL. JUN 86
UNCLASSIFIED VLSI-MEMO-86-327 N00044-80-C-0622 F/G 9/5 NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

(12)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 86-327
June 1986

METHODOLOGY VERIFICATION OF HIERARCHICALLY DESCRIBED VLSI CIRCUITS

Isaac L. Bain and Lance A. Glasser

Abstract

The standard approach to master the complexity of designing VLSI systems is to adopt a set of rules that, when respected, are conducive to correct implementations. Any such collection of rules can be called a design methodology. Most of the effort in computer-aided VLSI methodology verification has been traditionally concentrated on geometrical DRC. This paper describes a program that checks circuit conformity to other kinds of rules. This is done at the transistor level, and most of the rules are user-selected. Two related issues are also discussed: the description of digital MOS circuits using wiring operators; and the formal description of methodologies by the designer.

86 9 8 9

Microsystems Research Center Room 39-321
Massachusetts Institute of Technology
Cambridge Massachusetts 02139
Telephone (617) 253-8138

Acknowledgements

To be published in IEEE Trans. Computer-Aided Design. This work was supported in part by the National Science Foundation under grant number ECS-8118160, in part by the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622, and in part by a graduate fellowship from the Grass Instrument Company.

Author Information

Bain, current address: Alameda Itu, 890, #2, San Paulo, Brazil SP 01421; Glasser: Research Laboratory of Electronics and Department of Electrical Engineering and Computer Science, MIT, Room 36-880, Cambridge, MA 02139, (617) 253-4677.

METHODOLOGY VERIFICATION OF HIERARCHICALLY DESCRIBED VLSI CIRCUITS

Issac L. Bain

and

Lance A. Glasser

Department of Electrical Engineering and Computer Science

and the

Research Laboratory of Electronics

Massachusetts Institute of Technology

Cambridge, Massachusetts

ABSTRACT

The standard approach to master the complexity of designing VLSI systems is to adopt a set of rules that, when respected, are conducive to correct implementations. Any such collection of rules can be called a design methodology. Most of the effort in computer-aided VLSI methodology verification has been traditionally concentrated on geometrical DRC. This paper describes a program that checks circuit conformity to other kinds of rules. This is done at the transistor level, and most of the rules are user-selected. Two related issues are also discussed: the description of digital MOS circuits using wiring operators; and the formal description of methodologies by the designer.

This research was supported in part by the National Science Foundation under grant number ECS-8118160, in part by the Advanced Research Projects Agency under contract number N00014-80-C-0622, and in part by a graduate fellowship from the Grass Instrument Company.

METHODOLOGY VERIFICATION OF HIERARCHICALLY DESCRIBED VLSI CIRCUITS

1. Methodologies in VLSI Design

Typical VLSI circuits are very complex systems. Accordingly, their design is directed by a wide variety of considerations. The standard approach to mastering that complexity is to discipline the design process by adopting a number of rules that, taken together, are conducive to correct designs. Such a set of a priori selected rules, of which geometrical design rules are one aspect, is commonly referred to as a "methodology."

Different methodologies are better suited to achieving different goals, but all share a basic characteristic: they help isolate the designer from the details, allowing the design effort to be concentrated on higher-level abstractions. In that sense, the rules that constitute a methodology can be regarded as a syntax for the creation of circuits. An analogy is frequently made between VLSI circuits and large computer programs, with transistors corresponding to machine-language instructions and circuit cells to small routines; in that scenario a methodology can be thought of as corresponding to the syntactic requirements of a high-level language.

Methodology rules can be categorized according to the type of circuit information they refer to. TOPOLOGICAL RULES restrict the ways in which elements can be interconnected; they prescribe the configurations of pullups, pulldowns, pass structures, restoring logic gates, and more complex modules. TIMING RULES restrict the ways in which the designer can impose precedence of certain events over others to control the dynamics of the circuit; statements about clock waveforms and pre-charging of elements fall in this class. ELECTRICAL RULES establish limits on capacitances, resistances and other electrical properties of the various components of the circuit. LAYOUT RULES refer to geometrical shapes and dimensions on the mask. SPECIAL RULES account for useful special cases not predicted by the other categories; a special rule might, for instance, legalize a certain topology in a situation that would otherwise exclude it. (For example, in nMOS technology, part of the input protection circuitry includes an enhancement mode transistor with its gate and source shorted. This topology is only reasonable on chip inputs.) Rules belonging to the same class do not necessarily stem from similar considerations. Layout rules for nMOS technology, for instance, often include both design rules and pullup-to-pulldown ratio restrictions (beta ratios); the former are process-derived whereas the latter are related to the size of the noise margins.

As an illustration, the methodology presented by Mead and Conway in [1] can be informally summarized as follows: TOPOLOGICAL - depletion pullups, enhancement pulldowns and pass transistors, at most one threshold drop; TIMING - two-phase non-overlapping clocks; LAYOUT - 4:1 and 8:1 beta ratios, attendant lambda design rules; SPECIAL - superbuffers. As another example, the domino methodology proposed in [2] is characterized, in part, by p-type pullups, n-type pulldowns, and inverting buffers at the output of gates (topological); pre-charge of gate outputs (timing); inverting last stage (special).

2. A Methodology-Rule Checker

Among the advantages of observing a limited, well-defined set of rules is the possibility of having compliance verified by machine, either after the design is complete or as it proceeds in an interactive manner. The remainder of this paper describes CHECK-ME, a program that performs checking of methodology rules other than geometrical "design rules." (The important problem of DRC has been the object of most of the effort in computer-aided VLSI syntax verification [3,4], and even special-purpose hardware has been proposed for that end [5].) The information below summarizes the more complete documentation contained in [15].

A number of programs exist that execute some non-DRC pre-simulation checking; they do so either in parallel with DRC, as in [6], or after it, as in [7], and therefore require mask-level information.

The user almost invariably has no say in determining the categories of rules to be checked or the specific form they are to take. The main purpose of CHECK-ME is to evaluate in practice the concept of syntactic analysis of a circuit by providing a tool for interactively verifying compliance to a user-selected methodology, before any layout information is generated. In the process, two related issues are addressed: the description of digital MOS circuits using wiring operators; and the formal description of methodologies by the designer. Both issues are expanded in the next sections.

3. Circuit Description Language

Most of the rules of any given methodology involve the functionality of the circuit components to some extent. For example, a rule that outlaws a certain type of transistor in pullups assumes that there is a way to tell whether or not a particular device is being used as a pullup. The circuit description language of any methodology checker must, therefore, capture the designer's functional intent to some degree. Figure 1 illustrates that the inherent ambiguity of net-list languages of the kind used in most simulators makes them inadequate for that purpose. On the left side of the figure is a three transistor network. On the right side of the figure are two different interpretations of the functionality of the same basic topology. Without some indication of the designer's intent, it is impossible to tell whether or not the right-hand ground connection is a bug because it grounds the output of a pass transistor, or correct because is completes the NOR gate. There are many such examples in circuit design. For instance, one can not tell, by looking at the schematic, whether a pair of cross-coupled inverters is a latch or a regenerative amplifier. It can be either depending on the larger circuit context and the designer's intent.

3.1 *Wiring Operators: Concepts*

An input language based on WIRING OPERATORS has been chosen as the means for the user to express functionality while describing a circuit. The concept of wiring operators as an algebra for the description of electrical networks was first proposed in [8], and constituted the theoretical innovation of the analysis program MARTHA [9]. For the purposes of the methodology checker, the technique was generalized as suggested in [8] to accommodate multi-terminal digital MOS circuits.

Both the domain and image of these operators consist of external descriptions of subnetworks. The description of each subnetwork S is finite, and is considered external in the sense that it is the only information about S available to any wiring operator to which S is given as argument. The term "module" is now used to stand for both a subnetwork and its external description. Wiring operator w applied to modules A, B, C yields the description of module D obtained by connecting A, B, C in a certain manner, specified by w. For instance, operator WS connects its arguments in series, and WP, in parallel.

The wiring operator scheme has the pleasant side effect of encouraging a hierarchical design style. The language does not demand that the user assign a name or number to every node and device of the circuit. It enables the checker to work incrementally, flagging rules violations as they occur rather than after the whole circuit is described. It prevents the designer from committing, or renders readily detectable, some basic semantic mistakes like the creation of an information "sink" (a gate with a floating output, for example). It also expedites the discovery of some basic syntactic errors like the shorting of power supplies. And finally, it exposes designers to a novel way of representing digital circuits.

The methodology checker's input language uses MOS transistors as the basic elements of circuits. For the purposes of the wiring operators, transistors and all modules built thereof are viewed as controlled transmission structures, an instance of which is represented in figure 2. The external description of each module conveys syntactic and functional information, consisting of (a) a list of its external pins (subdivided into three groups named "left", "control" and "right" pins) and their attributes; and (b) module parameters. If the module is a transistor, the external pins are its drain, gate and source, falling into the left, control and right groups respectively; otherwise, they are determined by the sequence of wiring operators that created the module. In any case, they are the only terminals reachable by wiring operators to which the module is given as argument. Pin attributes encompass many of the information-flow ideas contained in [10] as well as the concept of signal strengths proposed in [11]. A pin's attribute is a symbol: an encoding of the attributes required of all pins that can be legally connected to it. Module parameters capture global properties by carrying information about the module as a whole, such as an overall numerical value.

Figure 3 shows three examples of modules. The external description of 3(a) consists of the names of terminals P, Q and R, attributes and parameters, all generated by the program when the device is declared. Module 3(b) is created by the sequential application of two wiring operators: the first connects transistors in series and the second grounds its argument's left pin. The resulting module has no left pins, two control pins and one right pin; attributes and parameters in its external description inform, among other things, that T is an output terminal and that the whole structure can be used as an enhancement-mode pulldown. Nodes P and R can no longer be reached by any wiring operator, in particular the one 3(b) was given as argument during the creation of 3(c). The other argument similarly resulted from the application of wiring operators to a depletion-mode transistor. Module 3(c) has two left pins, no control pins and one right pin, and is perceived by the program as a restoring gate.

3.2 Wiring Operators: Syntax

The operators are implemented as LISP functions and all other features of the input language have a LISP-like syntax. The modules in figure 3, for example, could be instantiated using the following expressions:

```
3(a) :     (enh 8 2)

3(b) :     (wgnd (ws (enh 8 2) (enh 12 3)))

3(c) :     (whook (wvdd (wshort-cr (dep 4 10)))
                  (wgnd (ws (enh 8 2) (enh 12 3))))
```

Primitives that create transistors, like ENH and DEP, are accompanied by two numbers, the declared channel width and length in that order. Operator WS returns the series connection of its arguments, while WGND and WVDD connect their arguments' left pins to the power supplies. Operator WSHORT-CR should only be applied to transistors -- it connects the control and right pins. The arguments to WHOOK must be two, one with the characteristics of a pullup and the other with those of a pulldown -- WHOOK joins them to form a gate.

Two other primitives are provided to declare zero-threshold and p-type devices. Additional wiring operators include the following functions: parallel connection of modules; shorting of left and control pins; stacking of modules; cascading of modules; creation of routers (modules consisting exclusively of wires); superimposing with pin-to-pin connection; and generation of dual structures.

The DEFNET command is used to define subcircuits, parameterized or not. Its syntax resembles that of LISP function definition. Once defined, these subcircuits can be used as if they were basic elements. The statements below are another way of instantiating the module of figure 3(c):

```
          (defnet series (width1 width2)
               (ws (enh width1 2) (enh width2 3)))

          (defnet simple-pullup (z)
               (wvdd (wshort-cr (dep 4 (* z 4)))))

3(c):     (whook (simple-pullup 2.5) (wgnd (series 8 12)))
```

4. Description of Methodologies

Different design technologies, styles, and objectives call for the adoption of different sets of methodology rules. Consequently, it is desirable that the checking program allow the user to specify the methodology before the circuit is described. It is the authors' experience that offering designers a general methodology description language to do so is impractical. Such a language requires formalizing rules to a degree of rigor that makes this description system unattractive to most designers.

Another approach is taken by CHECK-ME. First, it offers a menu whose entries are generalizations of the most popular currently-used methodologies. The designer is then directed to tailor the chosen one by setting some variables. Finally, for the more sophisticated user, a LISP-like rule description language is provided that can be used to create a limited variety of new rules. This language essentially enables the designer to influence the interplay of pin attributes and module parameters: it provides a means to specify what conditions each wiring operator will test its arguments for, and how it will generate the return information. WITHIN-PRIMITIVE is the special statement for doing that. Applied to the name of a circuit-description primitive and a piece of LISP code, it makes that code a part of the body of commands that implements the primitive.

To illustrate, suppose we want to adopt a rule that legalizes more than one voltage drop in pass structures made of zero-threshold devices, provided that their shape factor is greater than 3. Under this rule, the configuration of figure 4(b) will be legal if transistor T2 has w/l > 3. The primitive for declaring a zero-threshold transistor is ZER; the wiring operator that cascades modules and that, acting on the subcircuits in 4(a), results 4(b), is WC. In light of that, enforcement of the rule can be achieved with the following statements:

```
(within-primitive  zer
   (if (greaterp (// W L) 3)  (setf (ATTRIBUTE CONTROL-PIN) 'type-x)))

(within-primitive  wc
   (if (not (equal (ATTRIBUTE (INPUT-PIN  MODULE-ON-THE-RIGHT)) 'type-x))
       (error ">>>> Illegal configuration: too many threshold drops")))
```

The commands above have been stylized for reading convenience. All upper-case symbols stand for more verbose functions and variables that, in the actual commands, make reference to the data structures that hold the modules' external descriptions. Assume the shape factor of transistor T2 in figure 4(a) is greater than 3. By virtue of the first command, when that device is declared, the attribute "type-x" is associated with terminal G. Other attributes are assigned in the usual fashion to all other pins. As any pin attribute, "type-x" is just an arbitrary string that synthesizes information-flow and signal-strength properties of a pin. When operator WC is called, a procedure is run that checks, for every pair of pins to be connected, if any rules are being violated. As a consequence of the second command above, that procedure will also test the new "type-x" condition. Whereas without it only pair D-F would have been legalized, now pair E-G is also deemed valid.

5. A Word on Implementation and Usage

The methodology checker is embedded in LISP and is supported by the Schema database [12]. The system runs on Symbolics 3600 Lisp Machines. In general terms, the implementation of the wiring operators is as follows: first, the arguments are instantiated as Schema "modules." Then, all rule-checking conditionals are evaluated. If no mistakes are detected, Schema topological commands are issued to update the database representation of the circuit. Lastly, the new external description is returned. Pin attributes and module parameters are typically atoms, but can be any LISP object. The pins themselves are represented by Schema "pathnames." User-defined subcircuits created with DEFNET are implemented as Schema "types." The concepts of Schema pathnames, modules and types evolved from the similar DPL pathnames, instances and types. Those are well documented in [13].

The example of figure 3(c) can illustrate the program's basic mechanisms. When operator WHOOK is applied to the inner modules, it retrieves the values of some of their overall parameters. Assuming the Mead and Conway depletion-load nMOS methodology discussed in the first section is being checked, WHOOK will be particularly interested in its arguments' ***shape***, ***lineage*** and ***function*** , whose values in our case are respectively 0.4, "depletion-resistor," "pullup" for the first module, and 2.0, "enhancement-path," "pulldown" for the second. The initial tests are built-in: Is the first argument a pullup and the second a pulldown? Having passed these, the methodology-dependent tests are next: Is the pullup a depletion resistor? Is the pulldown an enhancement path? Is the ratio of the shapes greater than or equal to 4? Having verified that all that is true, Schema commands are issued to connect both modules' right pins. Finally, the new external description is generated: parameter ***function*** is set to "restoring-gate;" parameter ***ratio*** is set to the value 5.0; since it is less than 8, the attributes of both pins Q and S are set to "requires-restored-level" (rather than "accepts-one-drop"), and the attribute of pin T is set to "restored-output;" the pathnames of pins Q, S and T are also returned.

Among the tools used at several points in the description/verification/ correction iterative process that constitutes the VLSI design effort, CHECK-ME can be viewed as a 'first pass' before the lower-level, more computationally-intensive analysis programs. Additionally, if those programs draw their input files either directly or indirectly from the contents of the Schema database, CHECK-ME serves as a simplified circuit data entry interface. People exposed for the first time to CHECK-ME tend to perceive its wiring operator input language as the most significant obstacle to fluency. The user will find out, however, that the investment of a short time for practice suffices to gain a working familiarity with the operators' usage. There are only twelve of them, and their functions hardly overlap. At any point during circuit description the choice of an operator is strongly influenced by the context.

Schema itself is evolving. The checker can already be used in conjunction with some of its utilities that enable viewing modules graphically and generating input files to SPICE. The graphic editing and manipulation capabilities provided by Schema hold promise for nicely integrating interactive programs such as CHECK-ME. In fact, in CHECK-ME's case graphics can be incorporated right from the rules menu on. Other features planned for Schema, such as database representation of behavioral descriptions of signals, may prove useful in extending the program to give stronger emphasis to non-topological aspects of methodologies. In short, as the Schema environment is augmented, the checker will benefit even more from relying upon this powerful VLSI design system.

References

[1] C. Mead and L. Conway, _Introduction to VLSI Systems_, Reading: Addison-Wesley, 1980.

[2] R. Krambeck, C. Lee and H. Law, "High Speed Compact Circuits with cMOS" _IEEE Journal of Solid State Circuits_, June 1982.

[3] H. S. Baird, "A Survey of Computer Aids for IC Mask Artwork Verification" _Proceedings of the IEEE International Symposium on Circuits and Systems_, 1977.

[4] T. E. Whitney, "A Hierarchical Design-Rule Checker." Master's Thesis, California Institute of Technology, 1981.

[5] L. Seiler, "Special Purpose Hardware for Design Rule Checking." _Proceedings of the Second Caltech Conference on VLSI_, (Pasadena, CA) pp. 197-216, January 19-21, 1981.

[6] S. C. Johnson, "Hierarchical Design Validation Based on Rectangles," _Proceedings of the MIT Conference on Advanced Research in VLSI_, (Cambridge, MA), pp. 93-100, January 25-27, 1982.

[7] C. M. Baker, "Artwork Analysis Tools for VLSI Circuits," Master's Thesis, Massachusetts Institute of Technology, 1980.

[8] P. L. Penfield, Jr., "Description of Electrical Networks Using Wiring Operators," _Proceedings of IEEE_, vol. 60, pp. 49-53, January 1972.

[9] P. L. Penfield, Jr., _MARTHA User Manual_. Cambridge, MA: M.I.T. Press, 1971.

[10] L. A. Glasser "The Syntactic Analysis of VLSI Systems Using Graphs," unpublished, (M.I.T. VLSI Memo No. 81-62), September 1981.

[11] J. P. Hayes, "A Logic Design Theory for VLSI," _Proceedings of the Second Caltech Conference on VLSI_, pp. 455-476, January 19-21, 1981.

[12] G. Clark and R. Zippel, "Schema: An Architecture for Knowledge Based Design," _International Conference on Computer-Aided Design 85_, pp. 50-52, Santa Clara CA, November 18-21, 1985.

[13] J. Batali, N. Mayle, H. Shrobe, G. Sussman, D. Weise "The DPL/Daedalus Design Environment," in _VLSI 81, Proceedings of the First University of Edinburgh Conference on VLSI_, pp. 183-192, (Edinburgh, UK), August 18-21, 1981.

[14] A. Vladimirescu and S. Liu, "The Simulation of MOS Integrated Circuits Using SPICE2," University of California, Berkeley,

October 1981.

[15] I. L. Bain, "Methodology Verification of Hierarchically Described VLSI Circuits," Master's Thesis, Massachusetts Institute of Technology, May 1984.

Figures

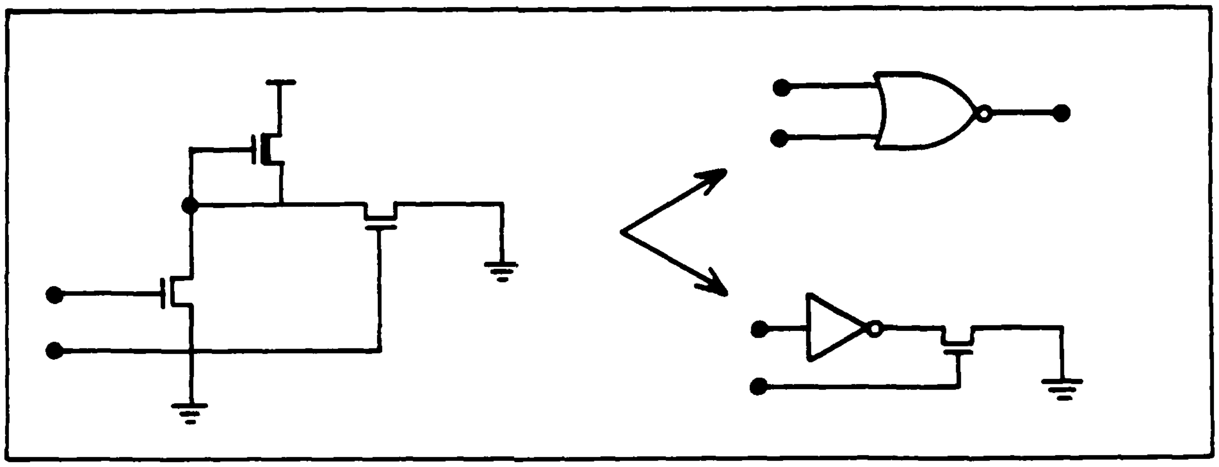


FIG 1

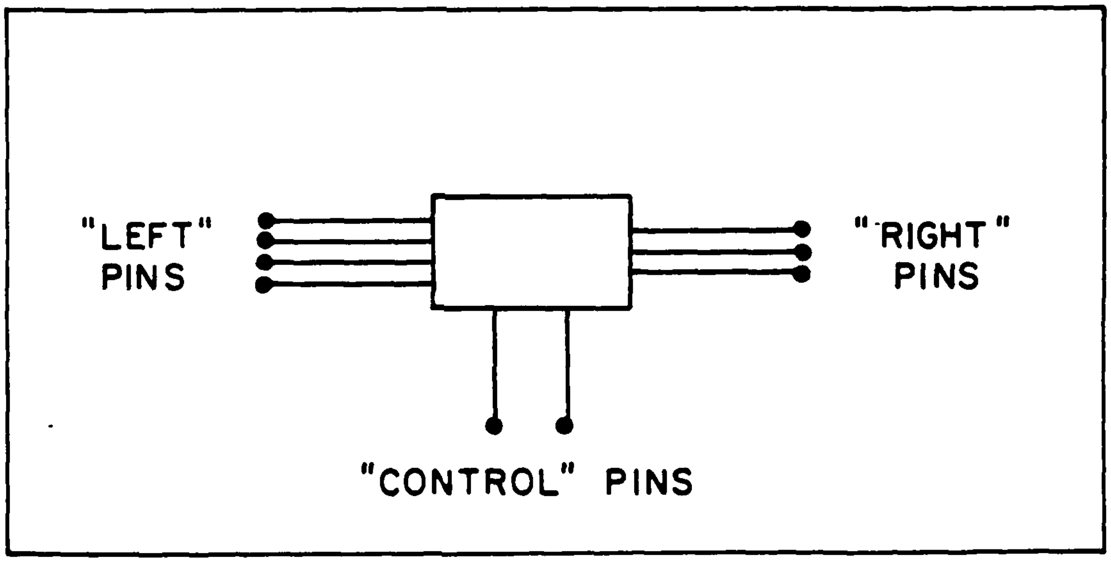


FIG 2

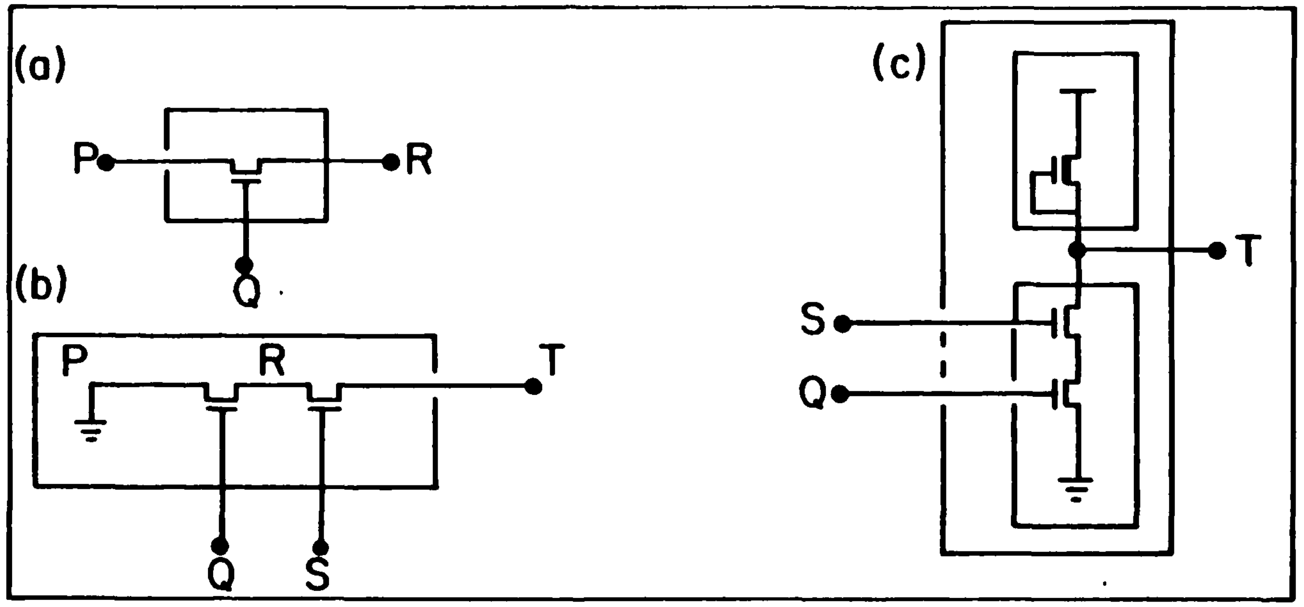


FIG 3

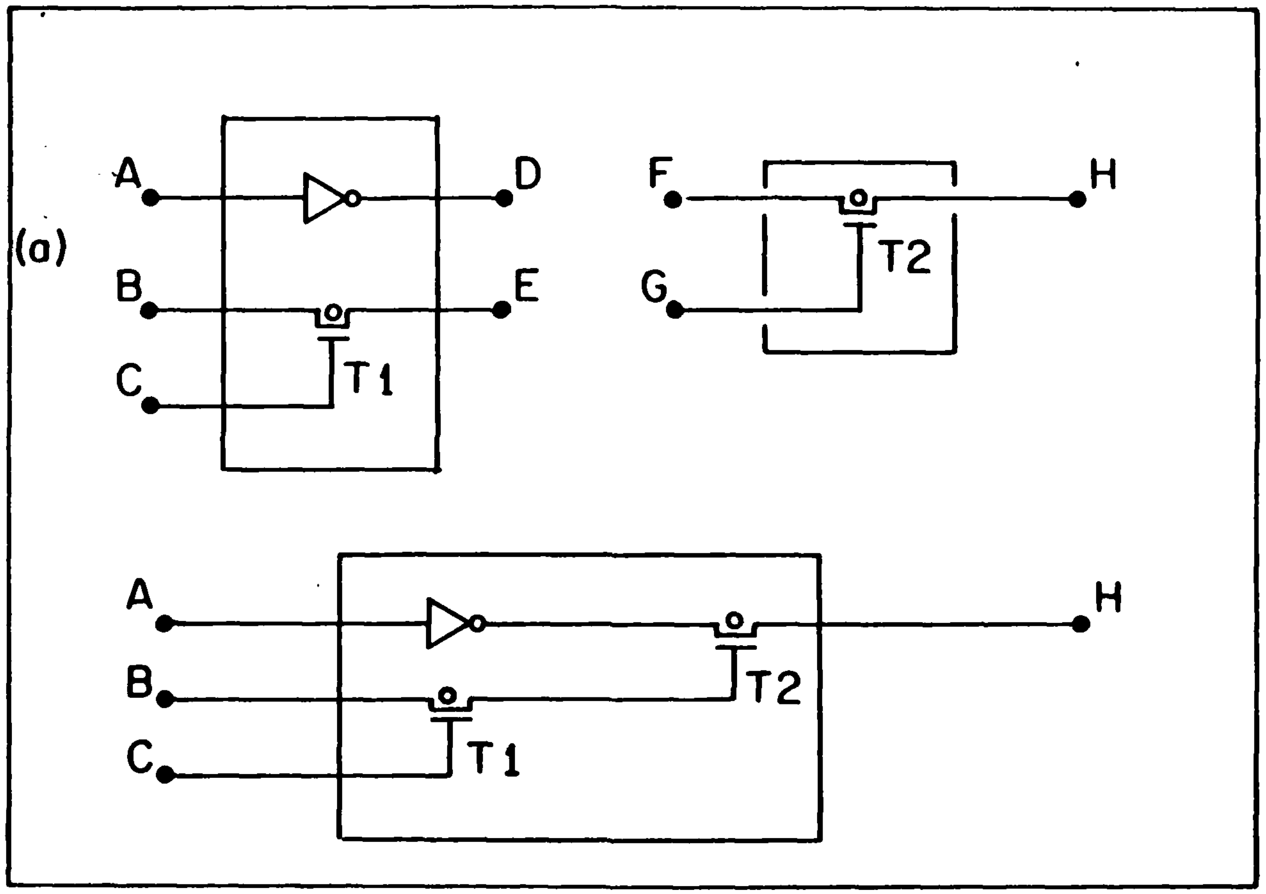


FIG 4

END
DTIC
10-86